Программирование

Python

Профилирование и оптимизация

Кафедра ИВТ и ПМ
ЗабГУ

2020

План

Прошлые темы

Профилирование

Оптимизация

Вычислительная сложность

Outline

Прошлые темы

- Как измерить время работы участка кода?

Прошлые темы

- Как измерить время работы участка кода?

```
import time
start_time = time.time()
// ...
execution_time = time.time() - start_time
```

Преждевременная оптимизация — корень всех зол

Дональд Кнут

Оптимизацию программы следует начинать с *узких мест* – частей кода, которые выполняются медленнее всего и потребляют больше всего ресурсов.

Определить эти места можно проанализировав алгоритм или измерив потребление ресурсов отдельными частями программы.

Outline

Профилирование

Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм), объём используемой памяти, число верно предсказанных условных переходов, число кэш-промахов и т. д.

Инструмент, используемый для анализа работы, называют **профилировщиком** или **профайлером** (англ. profiler). Обычно выполняется совместно с оптимизацией программы.

cProfile

cProfile - профилировщик кода Python. Измеряет число вызовов и время работы кода.

Во время запуска программы нужно совместно запустить cProfile:

```
python3 -m cProfile my_prog.py
```

cProfile построит таблицу в которой приведёт время выполнения отдельных функций и модулей, однако функции в таблице будут отсортированы по алфавиту.

Чтобы отсортировать функции по времени работы нужно указать профилировщику столбец, по которому будет произведена сортировка:

```
python3 -m cProfile -s tottime lab.py
```

https://docs.python.org/3/library/profile.html

cProfile. Пример

```
 s@laptop ~/Desktop/lab python3 -m cProfile -s cumtime lab.py
19395.05410680758
1955209.4819484616
         20001313 function calls (20001312 primitive calls) in 41.182 seco

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      2/1    0.000    0.000   41.182   41.182 {built-in method builtins.ex
        1    0.000    0.000   41.182   41.182 lab.py:2(<module>)
        1    0.005    0.005   20.670   20.670 lab.py:12(my_very_optimised_
     1000   10.698    0.011   20.665    0.021 lab.py:5(calc)
        1   10.688   10.688   20.510   20.510 lab.py:19(NOT_very_optimised
 20000000   19.789    0.000   19.789    0.000 {built-in method math.sin}
        2    0.000    0.000    0.002    0.001 <frozen importlib._bootstrap
        2    0.000    0.000    0.002    0.001 <frozen importlib._bootstrap
```

cProfile

Пояснения к выводу cPython

- `ncalls - число вызовов функции`
- `tottime - суммарное время выполнения функции (в секундах) без учёта выполнения всех функций вызываемых внутри данной.`
- `percall = totime / ncalls`
  `может быть показано как 0 потому, ели время меньше 0.001 секунды`
- `cumtime - суммарное время выполнения функции (в секундах) с учётом выполнения всех функций вызываемых внутри данной.`
- `percall = cumtime / ncalls`

cProfile. Разбор

- общее время работы программы с профилировщиком - 41.182 с.
- Функция которая выполнялась дольше всех в сумме вызовов: my_very_optimised_function (20.670 с.)
- Функция которая вызывалась чаще других: math.sin
- Пользовательская функция которая вызывалась чаще других: calc

Пример. Код.

```
from math import *
def calc():
        s = 0
        for i in range( 10000 ):
                s += sin(i)*10
        return s

def my_very_optimised_function(n):
        s = 0
        for i in range( n ):
                s += calc()
        return s

def NOT_very_optimised_function(n):
        s = 0
        for i in range( n ):
                for i in range(10):
                        s += sin(i)
        return s

print(  my_very_optimised_function(1000) )
print( NOT_very_optimised_function(1000000) )
```

cProfile

cProfile замеряет время работы всех функций и методов, в том числе системных, которые запускаются неявно.

Поэтому следует обращать внимание на имя файла, функции из которого были вызваны.
В примеры это lab.py, остальные функции, не из этой программы, можно не рассматривать.

Работа профайлера увеличивает время выполнения программы, поэтому его можно использовать для определения "узких мест"в программе, а не для измерения времени работы программы.

Outline

Оптимизация

Outline

Вычислительная сложность

Ссылки и литература

- Профилирование и отладка Python

Ссылки и литература

Ссылка на слайды

github.com/VetrovSV/Programming